```
//-----------------------------------------------------------------------------
// Temp_Lab.c
//-----------------------------------------------------------------------------
// Author: Baylor Electromechanical Systems
//
// Operates on an external 18.432 MHz oscillator.
//
// Target: Cygnal Educational Development Board / C8051F020
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//
// Controls the chip temperature by implementing DAC0 to control an external
// fan.  The fan varies speed according to a specified target temperature
// from the keypad. Output is display on the LCD
//

//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------

#include <c8051f020.h>                  // SFR declarations
#include <stdio.h>
#include <stdlib.h>

//-----------------------------------------------------------------------------
// 16-bit SFR Definitions for 'F02x
//-----------------------------------------------------------------------------

sfr16 TMR3RL   = 0x92;                  // Timer3 reload value
sfr16 TMR3     = 0x94;                  // Timer3 counter
sfr16 ADC0     = 0xbe;                  // ADC0 data
sfr16 DAC0     = 0xd2;                  // DAC0 data


//-----------------------------------------------------------------------------
// Global CONSTANTS
//-----------------------------------------------------------------------------

#define BAUDRATE     9600               // Baud rate of UART in bps
#define SYSCLK       18432000           // SYSCLK frequency in Hz
#define SAMPLE_RATE  5000               // Sample frequency in Hz
#define INT_DEC      256                // integrate and decimate ratio
#define command_length 2                // command length is 2 characters
// Lookup table for converting keycode to ASCII (for this lab, some keypad
// entries are disabled)
unsigned int keytab[4][4] ={{'1','2','3',0},
                            {'4','5','6',0},
                            {'7','8','9',0},
                            {0,'0',0,0}};



//-----------------------------------------------------------------------------
// Function PROTOTYPES
//-----------------------------------------------------------------------------

void SYSCLK_Init (void);
void PORT_Init (void);
void UART0_Init (void);
void ADC0_Init (void);
void Timer3_Init (int counts);
void ADC0_ISR (void);
int button_dn(void);
unsigned int scankey (void);
void delay_ms(int ms);

//-----------------------------------------------------------------------------
// Global VARIABLES
//-----------------------------------------------------------------------------
```

```
long result;                            // ADC0 decimated value
char input_str[3]= "";

//-----------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------

void main (void) {
    long temperature;                   // temperature in hundredths of a
                                        // degree C
    int temp_int, temp_frac;            // integer and fractional portions of
                                        // temperature
    int target_temp;
    int x;
    unsigned int rd1;


    WDTCN = 0xde;                       // disable watchdog timer
    WDTCN = 0xad;
    SYSCLK_Init ();                     // initialize oscillator
    PORT_Init ();                       // initialize crossbar and GPIO
    UART0_Init ();                      // initialize UART0
    Timer3_Init (SYSCLK/SAMPLE_RATE);   // initialize Timer3 to overflow at
                                        // sample rate
    ADC0_Init ();                       // init ADC
    AD0EN = 1;                          // enable ADC
    DAC0CN = 0x8C;                      // enable DAC0
    putchar (254);                      // LCD command
    putchar (0x01);                     // clear LCD
    EA = 1;                             // enable interrupts

    x=0;                                // string counter
    printf (" Please type\ntarget temp:");
    while (x<2)                         // 2 digit temp
    {
        if(button_dn())                 // check for key press
        {
            delay_ms(5);                // delay for debouncing
            rd1 = scankey();            // read keypad
            if(rd1 != 0)
            {
                putchar(rd1);           // send value to UART
                input_str[x]=rd1;       // add character to input_str
                x++;                    // increment counter
            }
            while(button_dn());         // check for key release
        }
        delay_ms(5);
    }
    delay_ms (1500);                    // delay a tad for 'asthetic' reasons
    target_temp = atoi (input_str);     // translate target temp
    putchar (254);                      // LCD command
    putchar (0x01);                     // clear LCD

    while (1)
    {
        EA = 0;                         // disable interrupts
        temperature = result;
        EA = 1;                         // re-enable interrupts

        // calculate temperature in hundredths of a degree
        temperature = temperature - 42380;
        temperature = (temperature * 100L) / 156;
        temp_int = temperature / 100;
        temp_frac = temperature - (temp_int * 100);

                                        // target temp + 1 degree
```

```
        if (temp_int >= target_temp + 1)
            DAC0 = 0x8000 ^ 32767;
        else
                                        // target temp + .50 degrees
            if ((temp_int == target_temp) && (temp_frac > 50))
                DAC0 = 0x8000 ^ 24000;
             else
                                        // target temp + .25 degrees
                if ((temp_int == target_temp) && (temp_frac > 20))
                {
                    DAC0 = 0x8000 ^ 30000;  // 'jump start it'
                    DAC0 = 0x8000 ^ 19000;
                }
                    else
                                        //target temp - .20 degrees
                        if ((temp_int == target_temp -1 ) && (temp_frac < 80))
                            DAC0 = 0;

        printf ("Temp = %+02d.%02d", temp_int, temp_frac);  // Display temp
        putchar (254);                  // LCD command
        putchar (0x02);                 // return home

    }
}

//-----------------------------------------------------------------------------
// Initialization Subroutines
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// This routine initializes the system clock to use an 18.432MHz crystal
// as its clock source.
//
void SYSCLK_Init (void)
{
   int i;                              // delay counter

   OSCXCN = 0x67;                      // start external oscillator with
                                       // 18.432MHz crystal

   for (i=0; i < 256; i++) ;           // XTLVLD blanking interval (>1ms)

   while (!(OSCXCN & 0x80)) ;          // Wait for crystal osc. to settle

   OSCICN = 0x88;                      // select external oscillator as SYSCLK
                                       // source and enable missing clock
                                       // detector
}

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports
//
void PORT_Init (void)
{
   XBR0    = 0x04;                 // Enable UART0
   XBR1    = 0x00;
   XBR2    = 0x40;                 // Enable crossbar and weak pull-ups
   P0MDOUT |= 0x01;                // enable TX0 as a push-pull output
   P1MDOUT |= 0x40;                // enable P1.6 (LED) as push-pull output

   P2MDOUT = 0xF0;                 // P2 u.n. push pull, lower-nibble input
   P2 = 0x0F;                      // upper nibble hi-imp, allowing input read
```

```
}

//-----------------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------------
//
// Configure the UART0 using Timer1, for <baudrate> and 8-N-1.
//
void UART0_Init (void)
{
   SCON0   = 0x50;                     // SCON0: mode 1, 8-bit UART, enable RX
   TMOD    = 0x20;                     // TMOD: timer 1, mode 2, 8-bit reload
   TH1    = -(SYSCLK/BAUDRATE/16);     // set Timer1 reload value for baudrate
   TR1    = 1;                         // start Timer1
   CKCON |= 0x10;                      // Timer1 uses SYSCLK as time base
   PCON  |= 0x80;                      // SMOD00 = 1
   TI0    = 1;                         // Indicate TX0 ready
}

//-----------------------------------------------------------------------------
// ADC0_Init
//-----------------------------------------------------------------------------
//
// Configure ADC0 to use Timer3 overflows as conversion source, to
// generate an interrupt on conversion complete, and to use left-justified
// output mode.  Enables ADC end of conversion interrupt. Leaves ADC disabled.
//
void ADC0_Init (void)
{
   ADC0CN = 0x05;                      // ADC0 disabled; normal tracking
                                       // mode; ADC0 conversions are initiated
                                       // on overflow of Timer3; ADC0 data is
                                       // left-justified
   REF0CN = 0x07;                      // enable temp sensor, on-chip VREF,
                                       // and VREF output buffer
   AMX0SL = 0x0f;                      // Select TEMP sens as ADC mux output
   ADC0CF = (SYSCLK/2500000) << 3;     // ADC conversion clock = 2.5MHz
   ADC0CF |= 0x01;                     // PGA gain = 2

   EIE2 |= 0x02;                       // enable ADC interrupts
}

//----------------------------------------------------------------------------
// Timer3_Init
//----------------------------------------------------------------------------
//
// Configure Timer3 to auto-reload at interval specified by <counts> (no
// interrupt generated) using SYSCLK as its time base.
//
void Timer3_Init (int counts)
{
   TMR3CN = 0x02;                      // Stop Timer3; Clear TF3;
                                       // use SYSCLK as timebase
   TMR3RL  = -counts;                  // Init reload values
   TMR3    = 0xffff;                   // set to reload immediately
   EIE2   &= ~0x01;                    // disable Timer3 interrupts
   TMR3CN |= 0x04;                     // start Timer3
}

//-----------------------------------------------------------------------------
// Interrupt Service Routines
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// ADC0_ISR
//-----------------------------------------------------------------------------
//
```

```
// ADC0 end-of-conversion ISR
// Here we take the ADC0 sample, add it to a running total <accumulator>, and
// decrement our local decimation counter <int_dec>.  When <int_dec> reaches
// zero, we post the decimated result in the global variable <result>.
//
void ADC0_ISR (void) interrupt 15
{
   static unsigned int_dec=INT_DEC;      // integrate/decimate counter
                                         // we post a new result when
                                         // int_dec = 0
   static long accumulator=0L;           // here's where we integrate the
                                         // ADC samples

   AD0INT = 0;                           // clear ADC conversion complete
                                         // indicator

    accumulator += ADC0;                 // read ADC value and add to running
                                         // total
   int_dec--;                            // update decimation counter

   if (int_dec == 0) {                   // if zero, then post result
      int_dec = INT_DEC;                 // reset counter
      result = accumulator >> 8;
      accumulator = 0L;                  // reset accumulator
   }
}

//-----------------------------------------------------------------------------
// Local Functions
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// button_dn
//-----------------------------------------------------------------------------
//
// Function: test keypad for the presence of a key press.
// Return:   1 if keypress; 0 otherwise.

int button_dn()
{
    int tmp;
    tmp = (P2 & 0x0F)^0x0F;              // read P2.3->P2.0 and XOR output

    if(tmp)                              // if button is depressed, tmp != 0
        return 1;
    else
        return 0;
}

//-----------------------------------------------------------------------------
// scankey
//-----------------------------------------------------------------------------
//
// Function: read keypad and convert keypress into equiv. ASCII code.
// Return: ASCII equivalent of pressed key's label.

unsigned int scankey(void)
{
    int row = 0;
    int col = 0;
    int k,j;

    P2 = 0x0F;                           // set data register
    P2MDOUT = 0xF0;                      // drive P2.3->P2.0 as output
    delay_ms(10);                        // let drive signals settle

    row = (P2 & 0x0F)^0x0F;              // read P2.3->P2.0 and XOR output
```

```
    delay_ms(2);

    if(row == 0)
        return 0;                       // no closure detected

    P2 = 0xF0;                          // set data register
    P2MDOUT = 0x0F;                     // drive P2.7->P2.4 as output
    delay_ms(2);                        // let drive signals settle

    col = (P2 & 0xF0)^0xF0;             // P2.7->P2.4 and XOR output
    col = col >> 4;                     // move hi nibble to lo nibble

    if(col == 0)
        return 0;                       // no closure detected

    P2 = 0x0F;                          // set data register
    P2MDOUT = 0xF0;                     // drive P2.3->P2.0 as output
    delay_ms(2);                        // let drive signals settle

    switch(row)                         // convert 1-of-4 to binary
        {
          case 1:   j = 0; break;
          case 2:   j = 1; break;
          case 4:   j = 2; break;
          case 8:   j = 3; break;
          default:  return 0;
        }

    switch(col)                         // convert 1-of-4 to binary
        {
          case 1:   k = 0; break;
          case 2:   k = 1; break;
          case 4:   k = 2; break;
          case 8:   k = 3; break;
          default:  return 0;
        }

    return keytab[j][k];    // return the ASCII value
}

//------------------------------------------------------------------------------
// delay_ms
//------------------------------------------------------------------------------
//
// an approximate x ms delay

void delay_ms(int ms)
{
    int y;
    int z;
    for (y=1; y<=250; y++) for (z=1; z<= ms; z++);
}
```